Resource Allocation for Query Optimization in Data Grid Systems: Static Load Balancing Strategies

Shaoyi Yin, Igor Epimakhov, Franck Morvan, Abdelkader Hameurlain

Resource Allocation for Query Optimization in Data Grid Systems: Static Load Balancing Strategies

Shaoyi Yin, Igor Epimakhov, Franck Morvan, and Abdelkader Hameurlain

IRIT Laboratory, Paul Sabatier University, France
{yin,epimakhov,morvan,hameurlain}@irit.fr

Abstract. Resource allocation is one of the principal stages of relational query processing in data grid systems. *Static* allocation methods allocate nodes to relational operations during query compilation. Existing heuristics did not take into account the multi-queries environment, where some nodes may become overloaded because they are allocated to too many concurrent queries. *Dynamic* resource allocation mechanisms are currently developed to modify the physical plan during query execution. In fact, when a node is detected to be overloaded, some of the operations on it will migrate. However, if the resource contention is too heavy in the initial execution plan, the operation migration cost may be very high. In this paper, we propose two load balancing strategies adopted during the static resource allocation phase, so that the workload is balanced at the beginning, the operation migration cost is decreased during the query execution, and therefore the average response time is reduced.

Keywords: Resource Allocation, Data Grid Systems, Query Optimization, Load Balancing.

1 Introduction

In recent years, large amount of scientific data have been produced and need to be shared by researchers from different organizations all over the world. Examples include the Large Hadron Collider (LHC) [1] at CERN and the Sloan Digital Sky Survey (SDSS) [2]. Data grid systems are designed to support such applications. With the objective of accessing and analyzing huge volume of data, the data grid systems rely on distributed, heterogeneous and autonomous computing resources [3]. On the one hand, a data grid system needs to perform query processing efficiently. On the other hand, a data grid system is characterized as large scale, heterogeneous and dynamic. Putting these two aspects together, many technical problems such as resource allocation become more challenging.

In a data grid system, there are different types of resources such as CPU, memory, storage and network. A node in the data grid system corresponds to a computer containing some of these resources. A user query is often issued on one of the nodes and is expressed using a declarative language such as SQL or OQL [4]. The query processor then transforms the query statement into an algebraic tree, called a logical query

plan, where nodes denote relational operations and edges represent data flows. Fig. 1 shows a query statement in SQL and its corresponding logical query plan.

At a given time, a node N of the data grid system which has access to a set of resources $R = \{r_1, r_2, \ldots, r_n\}$ receives a query Q which consists of a set of operations $\{o_1, o_2, \ldots, o_l\}$. Assume that R is collected during a preceding stage called resource discovery [5]. The problem of resource allocation is to assign one or more resources in R to each o_i such that the execution time of Q is minimized. Thereafter, we call the node N an *allocator*. Note that, the query type that we deal with is the Scan-Project-Join query. Obviously, the complexity of an exhaustive matching algorithm is exponential, so heuristics [6-9] have been proposed to solve this problem. However, most of these heuristics rely on the same principle: they first rank the operations according to certain criteria, then for each operation, they rank the available nodes and allocate the best ones to it. The criteria for ranking the operations, the criteria for ranking the nodes and the number of nodes to allocate are different in each heuristic method.

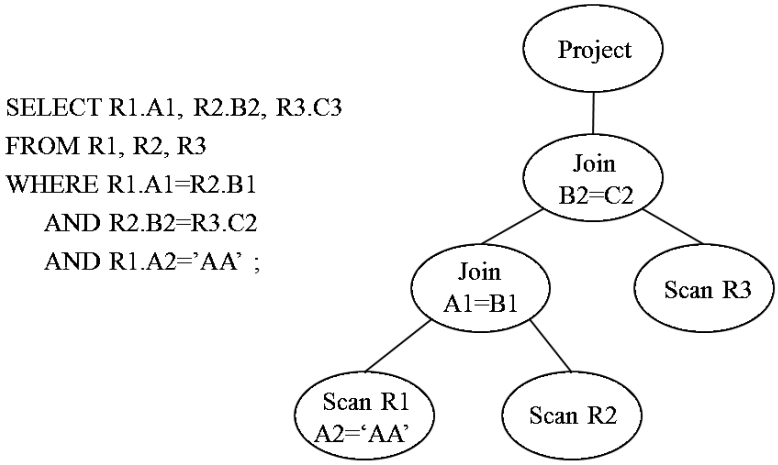

Fig. 1. A query statement and its logical query plan

These heuristics did not take into account the following constraints of the data grid system. *Constraint 1: Multi-queries* are treated by the same allocator. For each query, the allocator consumes the same list of candidate nodes, so the most powerful nodes are allocated to too many queries and become overloaded, while the least powerful nodes stay idle. *Constraint 2: Multi-allocators* co-exist in the data grid system (as shown in Fig. 2). The resources discovered for one allocator may also be discovered for other allocators. These resources will become overloaded if they are chosen by too many allocators.

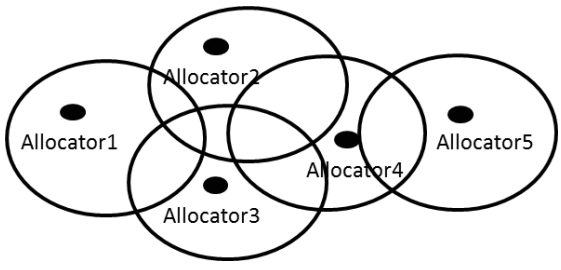

Fig. 2. Multi-allocators in a data grid system

The overloading problem caused by the above constraints is currently addressed by adding a resource reallocation phase (which is called *dynamic resource allocation*) during the query execution [10-15], that is, to move part of the work from overloaded resources to less loaded resources. The dynamic phase is very important, not only because the static allocation result may be under optimal, but also because in the data grid system, nodes may leave or enter at any time. However, the efficiency of the dynamic allocation is truly linked with the initial static allocation result. If the workload is already well balanced between nodes after the static resource allocation, much less work needs to be done during the dynamic phase, which is actually the objective of our paper.

In this paper, we propose two load-balancing strategies which aim at improving the physical execution plan generated by the static resource allocation, such that the average response time of the queries is reduced. They cannot replace the dynamic resource allocation phase, but they could make the latter more efficient. The principles of the proposed strategies are as follows:

- **Local Load Balancing (LLB):** virtually reserve resources of a node each time after it is allocated to an operation and virtually release the resources used by a query after it is finished;
- **Global Load Balancing (GLB):** Instead of allocating directly the best node to an operation, the algorithm first proposes K candidates and collects the current workload information of their resources by contacting them, then ranks the K nodes and returns the best one.

The rest of the paper is organized as follows. Section 2 presents a brief survey of the existing static and dynamic resource allocation methods in data grid systems. Section 3 describes the principles of our strategies and the corresponding algorithms. Section 4 evaluates the proposed strategies by combining them with existing resource allocation methods. Section 5 concludes the paper.

2 Related Work

The earliest work dealing with the resource allocation problem for query optimization in the data grid environment is the work of Gounaris[6]. It is a static resource allocation method. During the initiation phase, only one node is allocated to each operation. Then the algorithm increases the degree of parallelism for the most costly operation by allocating the most powerful nodes one by one to it. When the benefit of increasing a node to the operation is less than a threshold, the algorithm chooses the next most costly operation and increases the degree of parallelism for it. The iteration continues like this. When the chosen operation does not change any more, the resource allocation is finished. The load balancing is not addressed at all by this work.

Several other static resource allocation methods have been proposed [7-9]. The most recent one is called GeoLoc[9]. It has two main contributions. First, it defines an Allocation Space (AS) for each operation. It distinguishes two kinds of nodes: *data nodes* which contain the relation fragments and *computing nodes* which do not

con-tain any relation fragment. For a scan operation, the AS is defined to contain all the nodes storing the used relation fragments. For a join operation, the AS is defined to contain the nodes allocated to its input operations and the nodes geographically close to these nodes. Second, it takes into account the dependency between relational operations. It determines the degree of parallelism for each join operation according to the resource requirements driven by its input operations. For example, if the scan operations send 1000 tuples per second to a join operation, GeoLoc will allocate to that join operation just enough resources which can process 1000 tuples per second. Even though the load balancing problem is not explicitly addressed by the GeoLoc method, the usage of the AS decreases the resource contention. However, it is still possible that a same node is allocated for many queries simultaneously.

In the data grid system, due to the multi-queries and multi-allocators constraints, some nodes may become overloaded during the query execution, thus the average response time is increased. The current solution to this problem is to add a dynamic resource allocation phase which modifies the physical plan during the query execu-tion. There exist two main approaches: centralized and decentralized. In the central-ized approach, the workload status of the nodes is monitored by a dedicated resource broker [10-14]. In the decentralized approach, each node detects if it is over-loaded and makes autonomously the decision of moving operations from it to other nodes [15]. In the work [15], each relational operation is implemented as a mobile agent running on the allocated node, meaning that, it keeps track of its own status and can migrate to another node at any time. Thanks to a two-level cooperation mechanism between the autonomous nodes and autonomous operations, the workload is dynamically balanced among the grid nodes during query execution. The dynamic resource allocation phase is very important for query optimization, especially when there are node leavings and enters. However, the total migration cost could be high if the static resource allocation result is too imbalanced. In this paper, we aim at producing a balanced resource allocation result during the static phase, so that the migration cost of the dynamic phase is reduced, and therefore the average query response time is decreased.

3 Static Load Balancing Strategies

After the resource discovery phase, the allocator keeps the resource information (for example the size of memory, the CPU speed, the IO throughput and the network bandwidth) of each discovered node in a local table. This information is used as metadata during the resource allocation for the arriving queries. When the queries arrive frequently or even in batch, it is not realistic to update the resource information for each query due to the expensive communication cost. It means that, for a group (hundreds or thousands) of queries, the allocator consumes the same resource infor-mation table. This may introduce a kind of resource contention: some more powerful nodes may be allocated to too many concurrent queries while some less powerful nodes stay almost idle. Since the nodes are shared by the entire grid system, there is another kind of resource contention: a same node may be discovered for several different allocators and then become overloaded easily. In order to relieve these two kinds of resource contention, we propose in this section two load balancing strategies.

3.1 Local Load Balancing (LLB) Strategy

This strategy is proposed to solve the resource contention problem caused by the multi-queries constraint. The principle is to balance the workload between nodes by taking into account the local resource allocation history. In the resource information table RIT, the allocator maintains a counter for each resource type. Table 1 is a snapshot of the RIT, where NB_DM_X denotes the number of active demands for a resource type X and RC_X denotes the resource capacity on X. For example, if node N is allocated to a query which needs only I/O and network resources, we increase each of the counters $NB\_DM\_IO_N$ and $NB\_DM\_NET_N$ by one.

Table 1. Snapshot of the resource information table RIT

N_ID	RC_ RAM	RC_ CPU	RC_ IO	RC_ NET	NB_DM _RAM	NB_DM _CPU	NB_DM _IO	NB_DM _NET
205	15000	24000	19000	27000	10	5	8	10
192	8000	36000	12000	35000	6	12	3	5
…								

When the allocator allocates nodes for an operation, instead of using RC_X of each node to estimate the execution cost, it uses the currently available resource AR_X, which is computed using the following formula (where N is the node ID):

$$AR\_X_N = \frac{RC\_X_N}{NB\_DM\_X_N} \tag{1}$$

Table 2. Snapshot of the resource demand table RDT

Q_ID	N_ID	DM_RAM	DM_CPU	DM_IO	DM_NET
11	**205**	0	0	1	1
11	**192**	0	1	1	0
12					
…					

The allocator also maintains a table to register the resource demand of each treated operation. In this table, each tuple contains the query ID, node ID and the demand (DM_X, Boolean) for each resource type X. Each time after allocating a node to an operation, the allocator adds a tuple to this table. Each time a query is finished, the allocator decreases the demand counters in the resource information table RIT according to resource demand history related to the finished query. Finally, the history records related to that query are removed. Table 2 is a snapshot of the resource demand table RDT. For example, when query 11 is finished, we decrease each of the counters $NB\_DM\_IO_{205}$, $NB\_DM\_NET_{205}$, $NB\_DM\_CPU_{192}$ and $NB\_DM\_IO_{192}$ by one respectively, and we remove the first two tuples from table RDT.

Algorithm 1 describes the process of the maintenance of the resource information table and the resource demand table each time after allocating a node N to an operation O of a query Q. Algorithm 2 describes the process of maintenance each time after a query Q is finished.

Algorithm 1. VirtualResourceReservation()

```
INPUT: Query Q, Node N, resource demand DM_X for each X
BEGIN
    FOR each resource type X of node N DO
        IF DM_X > 0 THEN
         Increase the counter NB_DM_X_N by one in table RIT;
        END IF
      END FOR
     Insert the tuple (Q, N, DM_RAM, …) into table RDT;
END
```

Algorithm 2. VirtualResourceRelease()

```
INPUT: Query Q
BEGIN
  FOR each node N related to query Q in RDT DO
    FOR each resource type X of node N DO
      IF DM_X_N > 0 THEN
        Decrease the counter NB_DM_X_N by one in table RIT;
      END IF
    END FOR
  END FOR
  Delete the tuples related to Q from table RDT;
END
```

3.2 Global Load Balancing (GLB) Strategy

This strategy is proposed to solve the resource contention problem caused by the multi-allocators constraint. When choosing a node to allocate for an operation, the allocator proposes several candidate nodes according to the local resource information, and then it contacts these nodes to collect their workload status at the moment. Using the new resource information, the allocator ranks these candidate nodes and chooses the best one. Table 3 shows an example of candidate nodes list CNL after contacting them, where CNB_DM_X means the current number of demands for resource X registered by the node. The allocator estimates then the execution cost using the available resource capacity AR_X computed by the new formula:

$$AR\_X_N = \frac{RC\_X_N}{CNB\_DM\_X_N} \tag{2}$$

Table 3. Example of the candidate nodes list CNL

N_ID	CNB_DM _RAM	CNB_DM _CPU	CNB_DM _IO	CNB_DM _NET
39	12	6	9	10
267	8	3	6	7
…				

Algorithm 3 illustrates the steps of using GLB to select a node N for an operation O. Note that, the current workload information collected from the nodes is used only to choose the best node for an operation, but not to update the RIT table. The RIT table could be only modified by the LLB strategy so that the information is kept consistent.

Algorithm 3. NodeSelection()

```
INPUT: the resource information table RIT, an operation O
OUTPUT: the node N to be allocated to O
BEGIN
    Choose K candidate nodes using RIT;
    FOR each chosen node N DO
        Contact N and get CNB_DM_X_N for each resource X;
        Insert the collected information into CNL;
    END FOR
    FOR each node N in CNL DO
        Estimate the execution cost of O by adding N to it;
    END FOR
    Return the node N with the minimal estimated cost;
END
```

4 Performance Evaluation

We first measure the impact of LLB and GLB strategies on two existing static resource allocation methods: method of Gounaris [6] and GeoLoc method [9]. We then add the dynamic resource allocation phase to each method and measure again the impact of LLB and GLB. The dynamic allocation algorithm that we use is the work published in [15]. Our proposed load balancing strategies can be combined with other static or dynamic resource allocation methods, even though we choose only the above representatives for the performance evaluation.

For the experimentation, we use the grid simulator presented in [9], [15] with some extensions. The simulated data grid contains 2000 nodes that store 2000 relations. Each relation contains 5 equal fragments, each of which in turn is duplicated on 4 nodes. So in average we have 20 copies of different fragments on each node. To simulate the dynamicity of the grid system, 5% of the nodes quit the system at the 600^{th} millisecond and reenter at the 1200^{th} millisecond. The main parameters that we used for the simulation are listed in Table 4.

Table 4. System configuration and database parameters

	Parameter	Value
Node	CPU performance	100 - 10 000 MIPS
	I/O throughput	10 – 1000 Mb/s
	Memory amount	10 – 1000 MB
	Network connection bandwidth	10 – 1000 Mb/s
	Network connection latency	0.05s
Relation	Number of attributes	10
	Size of attribute	1 – 50 Bytes
	Cardinality of attributes	0.01 – 1
	Number of tuples	1000 – 11000
	Number of fragments	5
	Number of duplicates per fragment	4

Section 4.1 shows the impact of the proposed strategies on the static resource allocation, in terms of average query response time and optimization cost. Section 4.2 shows the impact of the proposed strategies on the static resource allocation combined with a dynamic resource allocation phase, in terms of average query response time and number of operation migrations. We also measured the effect of increasing the number of concurrent queries.

4.1 Impact of LLB and GLB on Static Resource Allocation

We take Gounaris' method and the GeoLoc method as two examples to examine our load balancing strategies. LLB and GLB strategies are evaluated in a gradual way, meaning that, we first measure the initial allocation method (named *Gouna/GeoLoc*), then the initial method combined with LLB (named *GouLLB/GeoLLB*), and finally the initial method combined with LLB and GLB (named *GouLLB+GLB /GeoLLB+GLB*).

Impact on Gounaris' Method. The average response time varying with the number of queries is shown Fig. 3. The speedup factor of the two strategies is illustrated in Fig. 4. Not surprisingly, we find that the speedup factor is more significant when there are more concurrent queries.

We measured also the average optimization cost of each query by varying the number of queries. The result can be found in Fig. 5. Interestingly, we see that the optimization time is not always longer when adding our load balancing strategies. This is because the query optimization contains several steps including mainly 1) generation of the logical plan, 2) selection of nodes for each operation and 3) generation of physical operators. When adding LLB or GLB, the first step does not change, the second step is slowed down, but the third step could be sped up or slowed down depending on the result of the second step. For example, when adding LLB strategy, fewer physical operators are generated during the third step, so the optimization time is shorter.

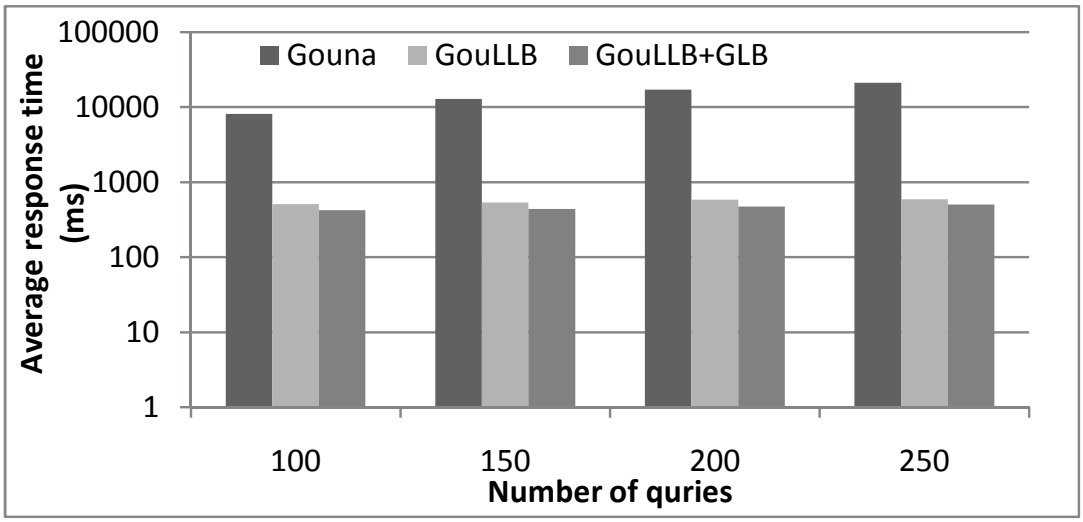

Fig. 3. Average query response time varying with the number of queries

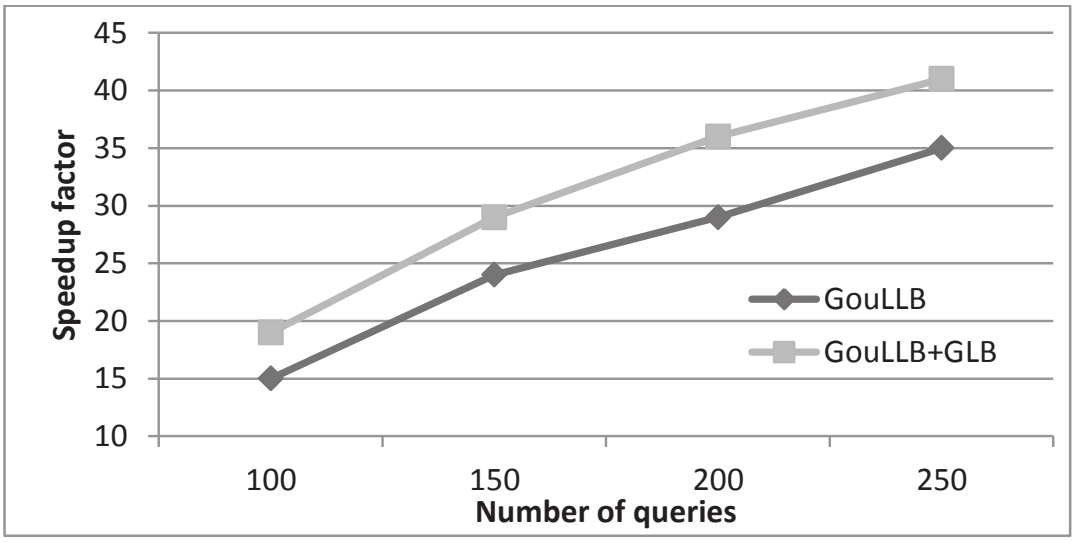

Fig. 4. Speedup factor of load balancing strategies varying with the number of queries

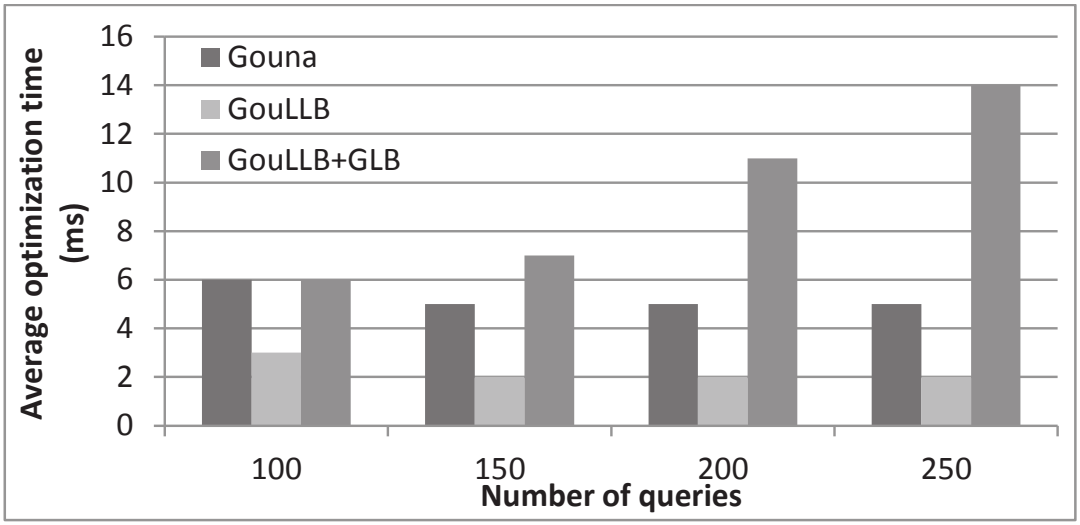

Fig. 5. Average optimization time per query varying with the number of queries

Impact on GeoLoc Method. The average response time and the speedup factor are shown in Fig. 6 and Fig. 7 respectively. The impact of the load balancing strategies is less significant than for Gounaris' method, because GeoLoc has already balanced the workload to some extent thanks to the using of Allocation Space. However, the same conclusion can be drawn: the speedup factor of LLB and GLB is more important when there are more concurrent queries. The average optimization time of each query is given in Fig. 8. We have three remarks: 1) the optimization time is not always increased by adding the load balancing strategies; 2) even when the optimization time is increased, the discrepancy is not very high; 3) compared to the total response time, the optimization cost is trivial.

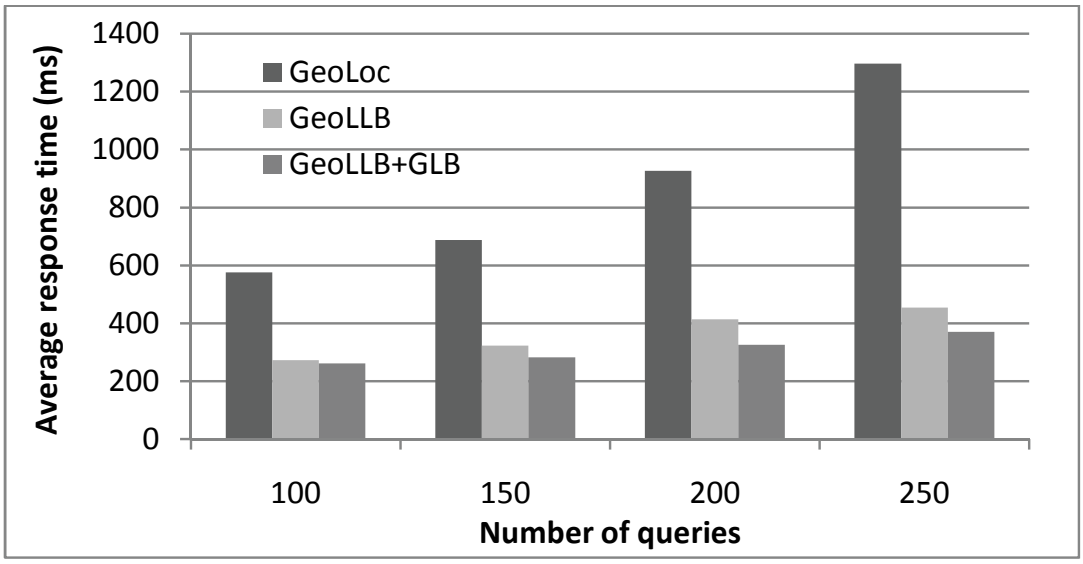

Fig. 6. Average query response time varying with the number of queries

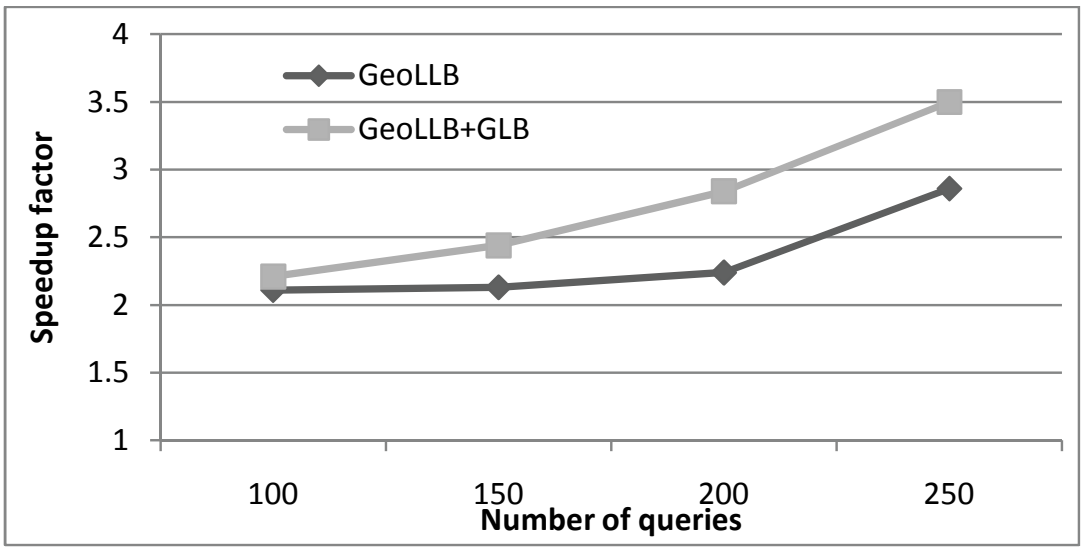

Fig. 7. Speedup factor of load balancing strategies varying with the number of queries

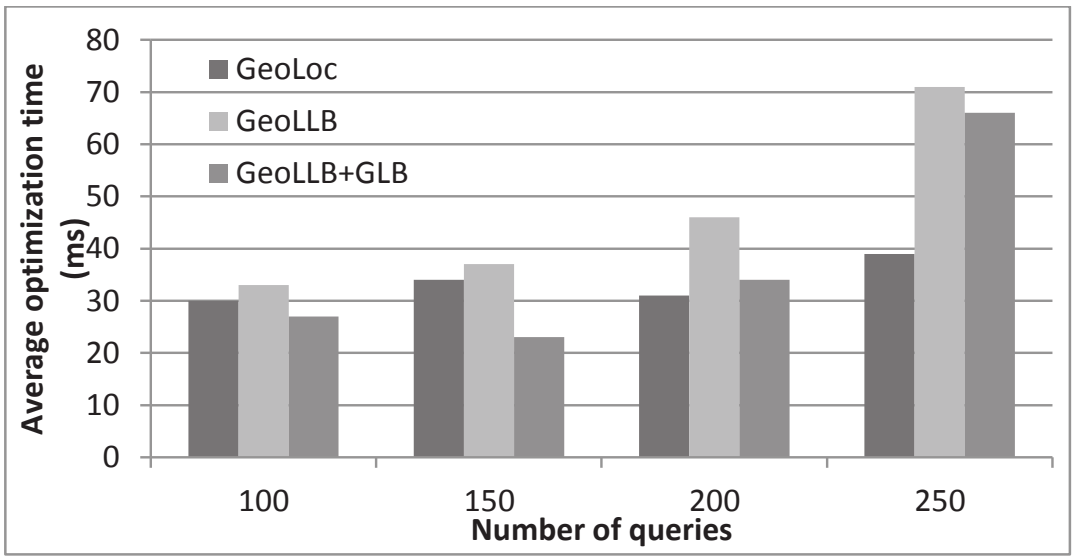

Fig. 8. Average optimization time per query varying with the number of queries

4.2 Impact of LLB and GLB on Dynamic Resource Allocation

We still take Gounaris' method and the GeoLoc method as two examples of static resource allocation methods. A dynamic resource allocation phase is added to each using the same mechanism published in [15]. We evaluate the impact of our load balancing strategies in terms of the average response time of a query and the number of operation migrations in the system during query execution. The notations of the methods are the same as in Section 4.1.

Impact on Gounaris' Method. The average response time and the speedup factor are shown in Fig. 9 and Fig. 10 respectively. The speedup factor is less high than doing only static allocation, but it is still remarkable. The total number of operation migrations during the query execution is given by Fig. 11. We can see that LLB and GLB have avoided many operation migrations, and this is one of the reasons why the average response time is reduced.

Impact on GeoLoc Method. The average response time and the speedup factor are shown in Fig. 12 and Fig. 13 respectively. The number of operation migrations is not shown due to space limitation. The conclusion is that, LLB and GLB have avoided most of the operation migrations during execution, and the average response time is reduced accordingly.

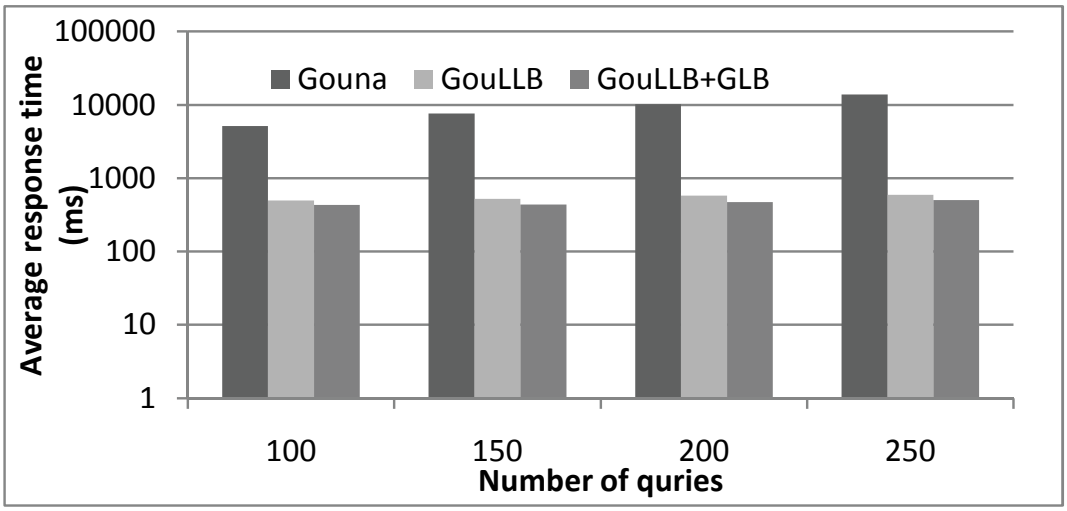

Fig. 9. Average query response time varying with the number of queries

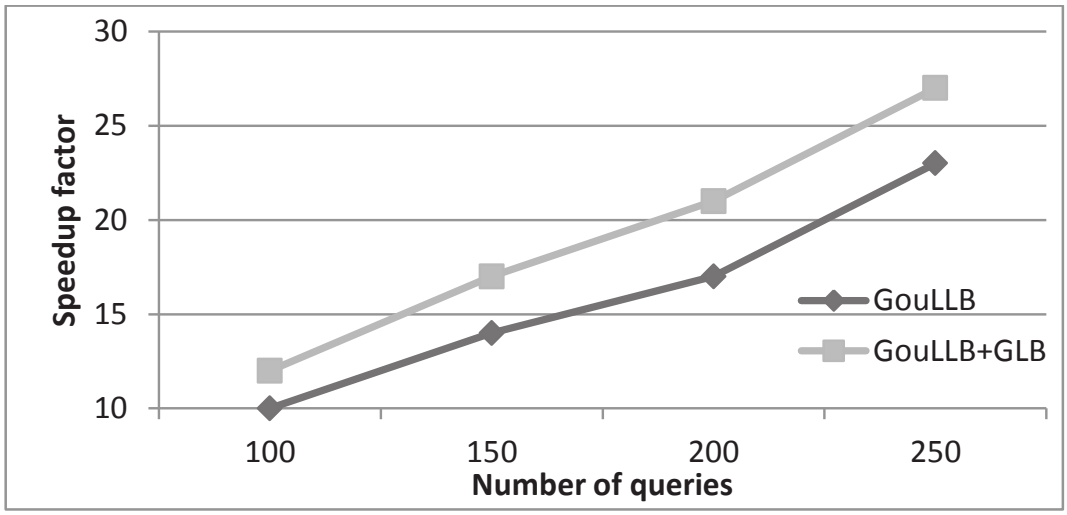

Fig. 10. Speedup factor of load balancing strategies varying with the number of queries

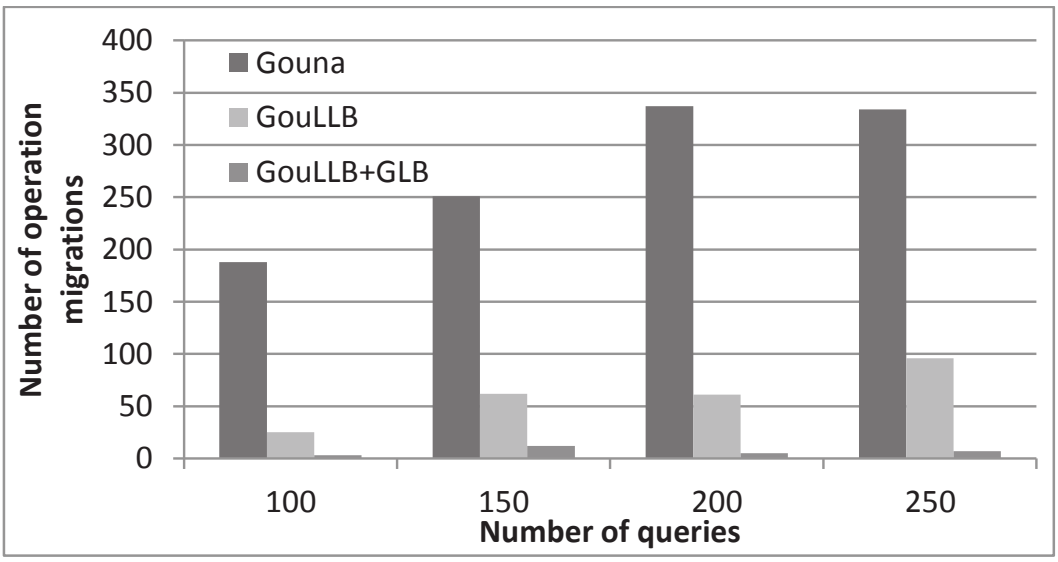

Fig. 11. Number of operation migrations varying with the number of queries

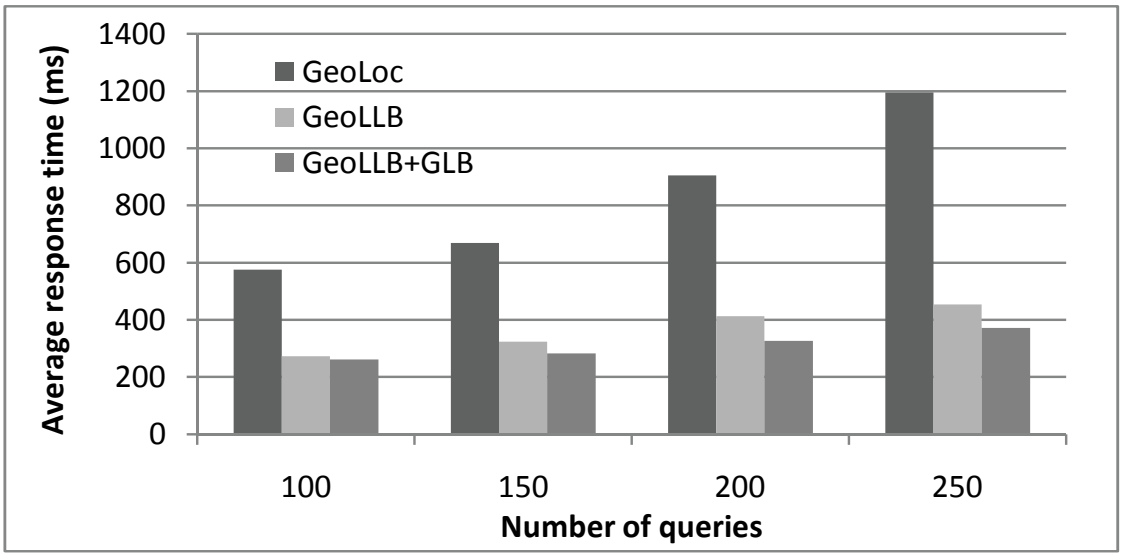

Fig. 12. Average query response time varying with the number of queries

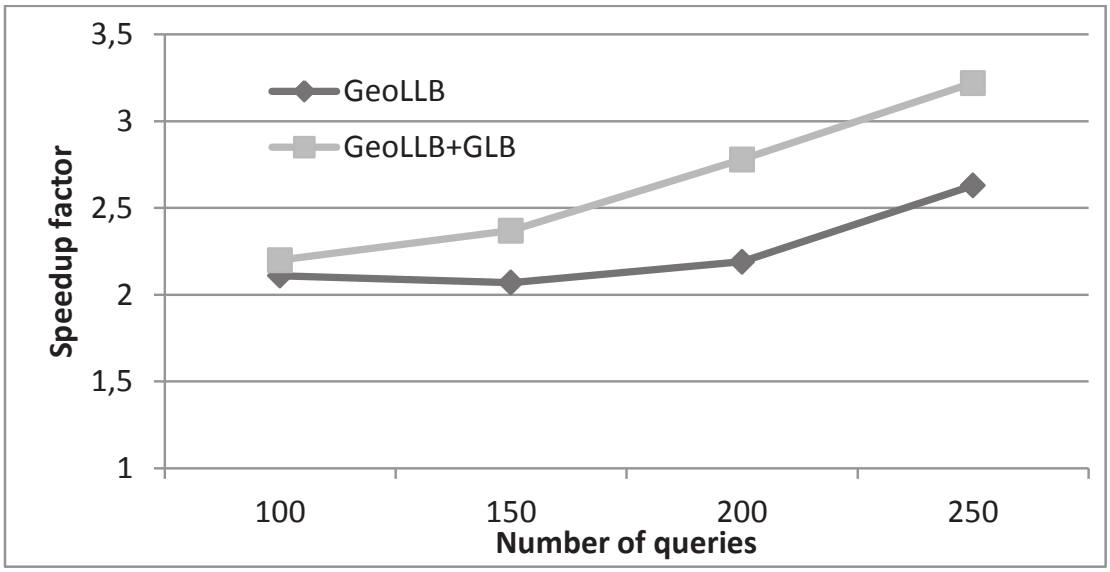

Fig. 13. Speedup factor of load balancing strategies varying with the number of queries

5 Conclusion

In this paper, we presented two static load balancing strategies during resource allocation for query optimization in data grid systems, in order to avoid the node overload situation, so that the average query response time is reduced. When combined with the dynamic allocation phase, they could reduce the operation migration cost during execution and therefore decrease the query response time accordingly.

The first strategy is called Local Load Balancing (LLB). It is designed to solve the resource contention problem caused by the multi-queries constraint of data grid systems. It makes fully use of the local resource allocation history to finally distribute the workload proportionally according to the capacity of the nodes. The second strategy is called Global Load Balancing (GLB). It is designed to solve the resource contention

problem caused by the multi-allocators constraint of data grid systems. It first chooses several candidate nodes for an operation using the local information, then contacts these nodes and collects their current workload status, and finally re-ranks these nodes to select the best one.

The result of the performance evaluation has shown the efficiency of the proposed strategies. For example, by integrating our LLB and GLB strategies with Gounaris' method, the query execution time is reduced by 10 to 40 times; by integrating them with the GeoLoc method, the query execution time is reduced by 2 to 4 times. The proposed strategies could be combined with other existing static and dynamic resource allocation methods in data grid systems. They decrease significantly the query response time, but they don't increase much the allocation cost.

References

1. http://lhc.web.cern.ch/lhc/
2. http://www.sdss.org/
3. Chervenak, A., et al.: The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets. Journal of Network and Computer Applications 23, 187–200 (1999)
4. Smith, J., Gounaris, A., Watson, P., Paton, N.W., Fernandes, A.A.A., Sakellariou, R.: Distributed Query Processing on the Grid. In: Parashar, M. (ed.) GRID 2002. LNCS, vol. 2536, pp. 279–290. Springer, Heidelberg (2002)
5. Krauter, K., et al.: A taxonomy and survey of grid resource management systems for distributed computing. Journal of Software: Practice and Experience 32, 135–164 (2002)
6. Gounaris, A., et al.: Resource scheduling for parallel query processing on computational grids. In: GRID (2004)
7. Soe, K.M., et al.: Efficient scheduling of resources for parallel query processing on grid-based architecture. In: Information and Telecommunication Technologies (2005)
8. Liu, S., Karimi, H.A.: Grid query optimizer to improve query processing in grids. Future Gener. Comput. Syst. 24, 342–353 (2008)
9. Epimakhov, I., et al.: GeoLoc: Robust Resource Allocation Method for Query Optimization in Data Grid Systems. In: DB&IS (2012)
10. Gounaris, A., et al.: Adaptive query processing and the grid: Opportunities and challenges. In: DEXA Workshops (2004)
11. Gounaris, A., et al.: Practical adaptation to changing resources in grid query processing. In: Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006 (2006)
12. Da Silva, V.F.V., et al.: An adaptive parallel query processing middleware for the grid. Concurrency and Computation: Practice and Experience 18(6), 621–634 (2006)
13. Avnur, R., Hellerstein, J.M.: Eddies: Continuously adaptive query processing. In: Proceedings of the SIGMOD Conference, pp. 261–272 (2000)
14. Patni, J., et al.: Load balancing strategies for grid computing. In: Proceedings of the 3rd International Conference on Electronics Computer Technology, ICECT (2011)
15. Epimakhov, I., et al.: Mobile Agent-based Dynamic Resource Allocation Method for Query Optimization in Data Grid Systems. In: International KES Conference on Agents and Multi-agent Systems – Technologies and Applications (2013)